Introduction to SDN

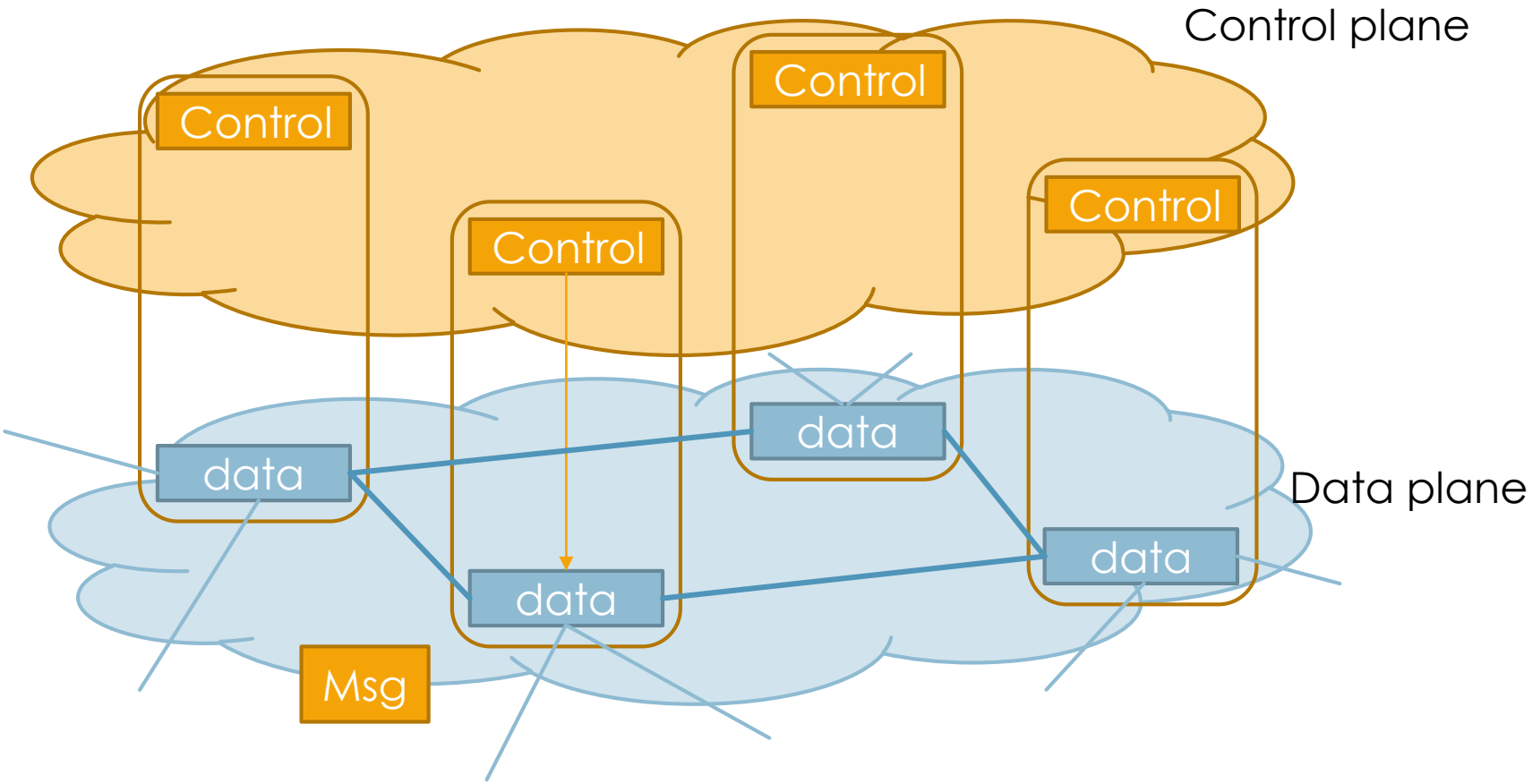
LI, RUNHUI

ANSR LAB, DEPT. OF COMPUTER SCI & TECH
THE CHINESE UNIVERSITY OF HONG KONG

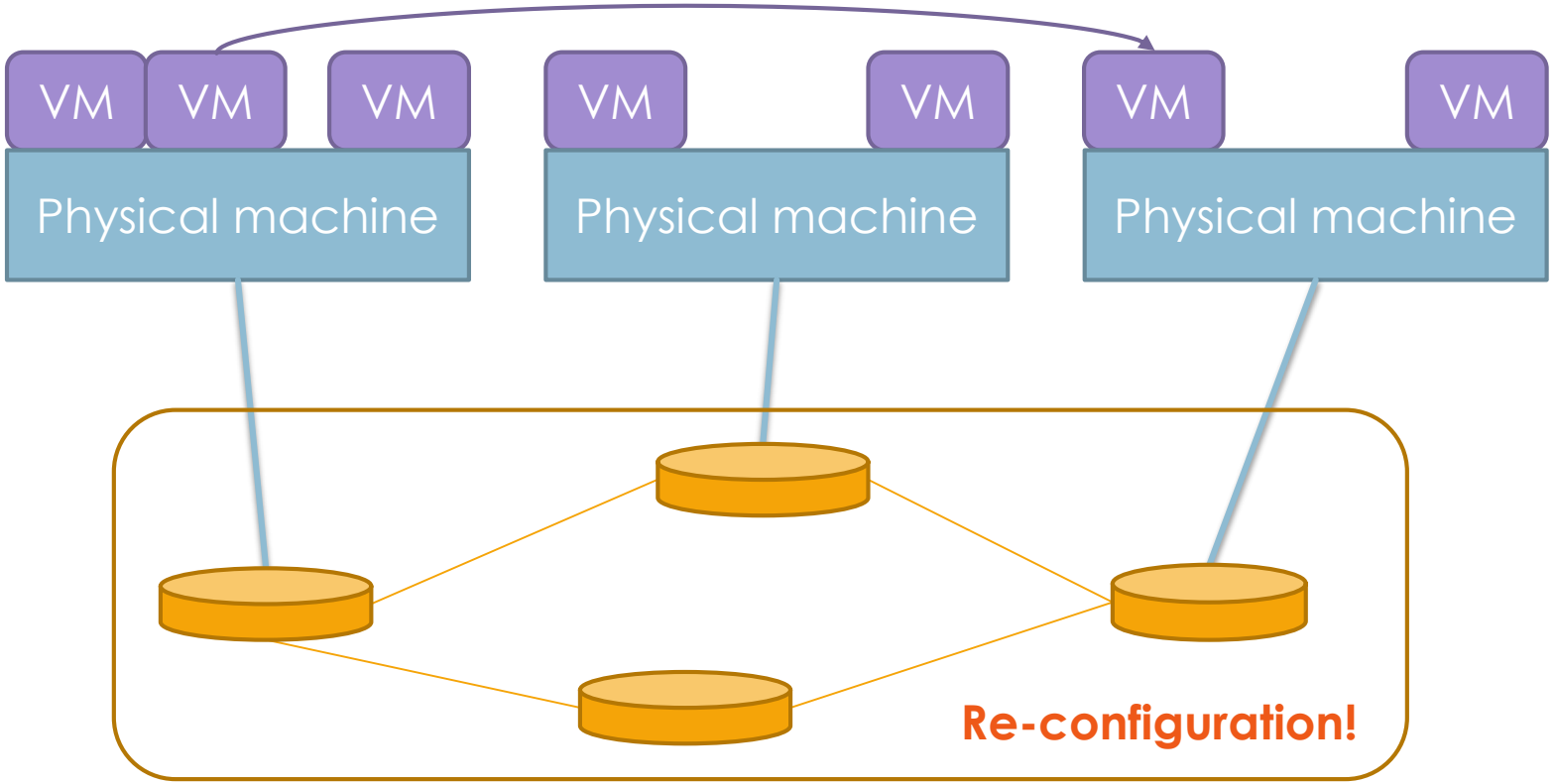
Roadmap

- SDN
  - Design principles
  - SDN infrastructure
  - OpenFlow: manage packet flow in SDN
- Mininet: SDN simulator
- Lessons learned

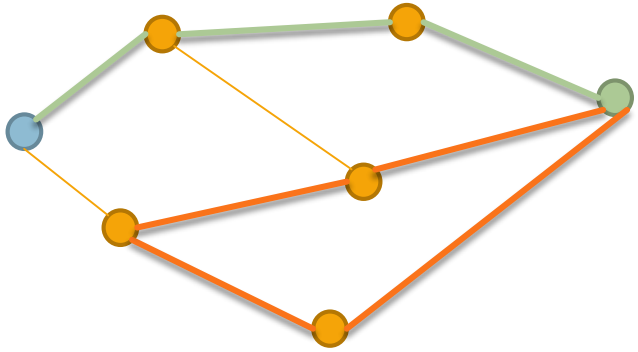
Traditional Network

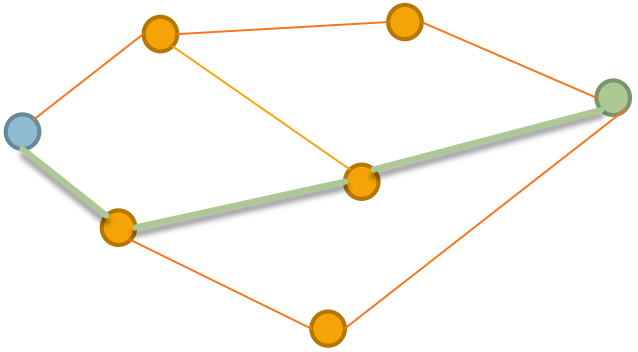


Example: VM Migration



Example





- Load balancing
  - Lack of **global view**
  - Lack of **centralized control**
- Security path
  - Lack of **flexible forwarding**

Management is Disaster!

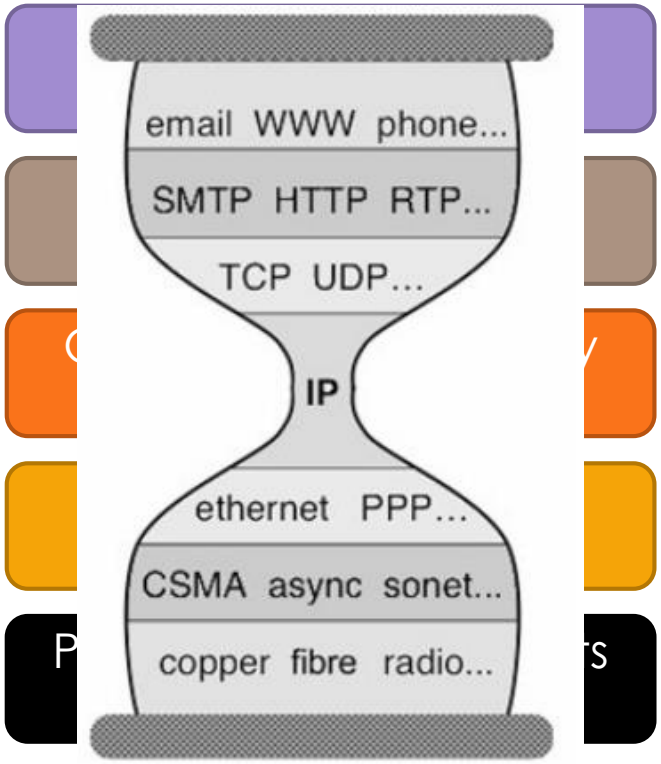
- Compute configuration for distributed devices
  - Forwarding tables
- Configure without communication guarantee
- Given network-level protocols

Motivation

- Traditional network:
  - Difficult to manage
  - (Re-)configuration is **disaster**
  - Bad control → performance, security issues
- Network increasingly **dynamic** and **complicated**
- What are we expecting
  - Open
  - Programmable

Lessons from Data Plane

- Data transmission is **NEAT**
- **Layering** in data plane
- Layering excellent **abstraction**
  - Decomposition
  - **Independent**
  - **Compatible**
- Control plane: **lack of *abstraction***



Network Control Problem

- **Distribution** to physical devices
  - Abstraction for distributed state
- **Configuration** of each physical device
  - E.g., forwarding tables, ACLs
  - Abstraction for simplified configuration
- **Forwarding** with given network-level protocol
  - Abstraction general forwarding model

What is desired abstraction?

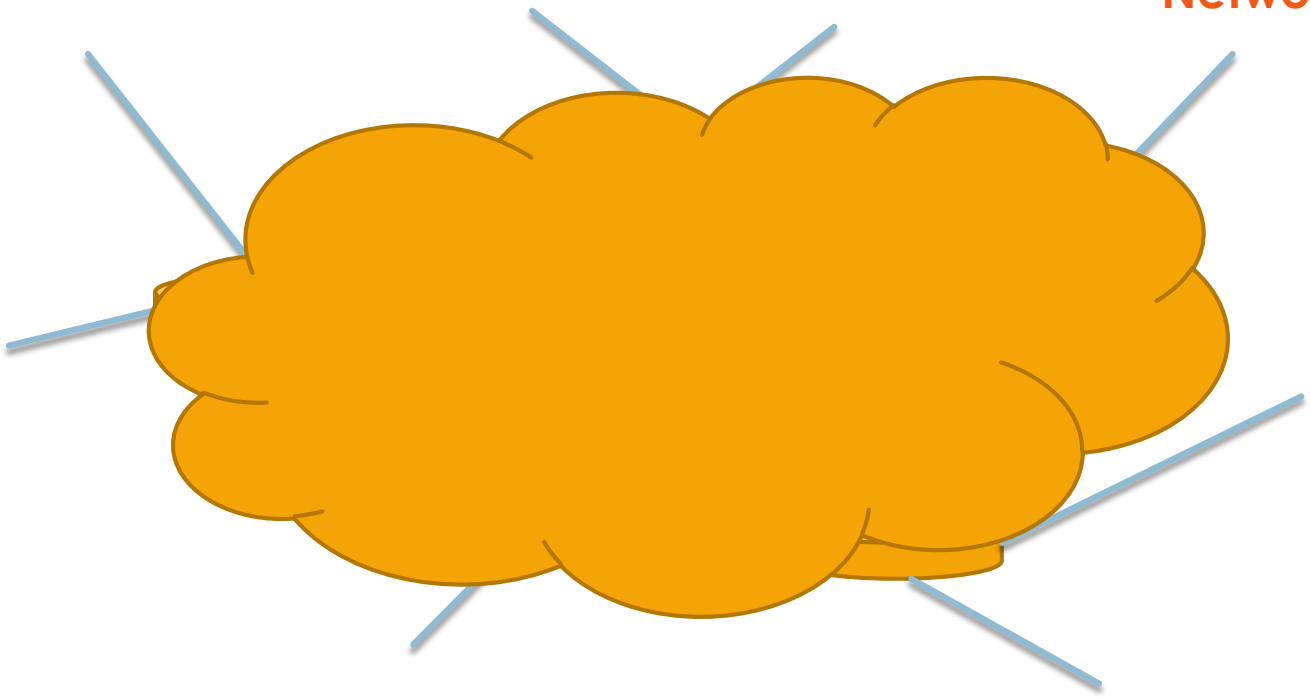
Distribution

- Shield distributed states
  - Focus on **mechanism**
- Abstraction: **Global network view**
  - Get network graph through API
- Control
  - ~~Distributed protocols~~ → Graph algorithm

Configuration

- **Control program**: desired behavior
- Do **NOT** need to concern the **implementation**
- What abstraction to control program

Example: Access Control



Configuration

- **Control program**: desired behavior
- Do **NOT** need to concern the **implementation**
- What abstraction to control program
- Simplified view
  - Enough and only enough to specify desire
  - Behavior = function (view)

Forwarding

- Given network-level protocol → **Flexible** forwarding model
- Should **NOT** constrain control program
  - Support **any type** of forwarding behavior
- Hide **details** of underlying hardware
  - Translate to configurations of networks elements

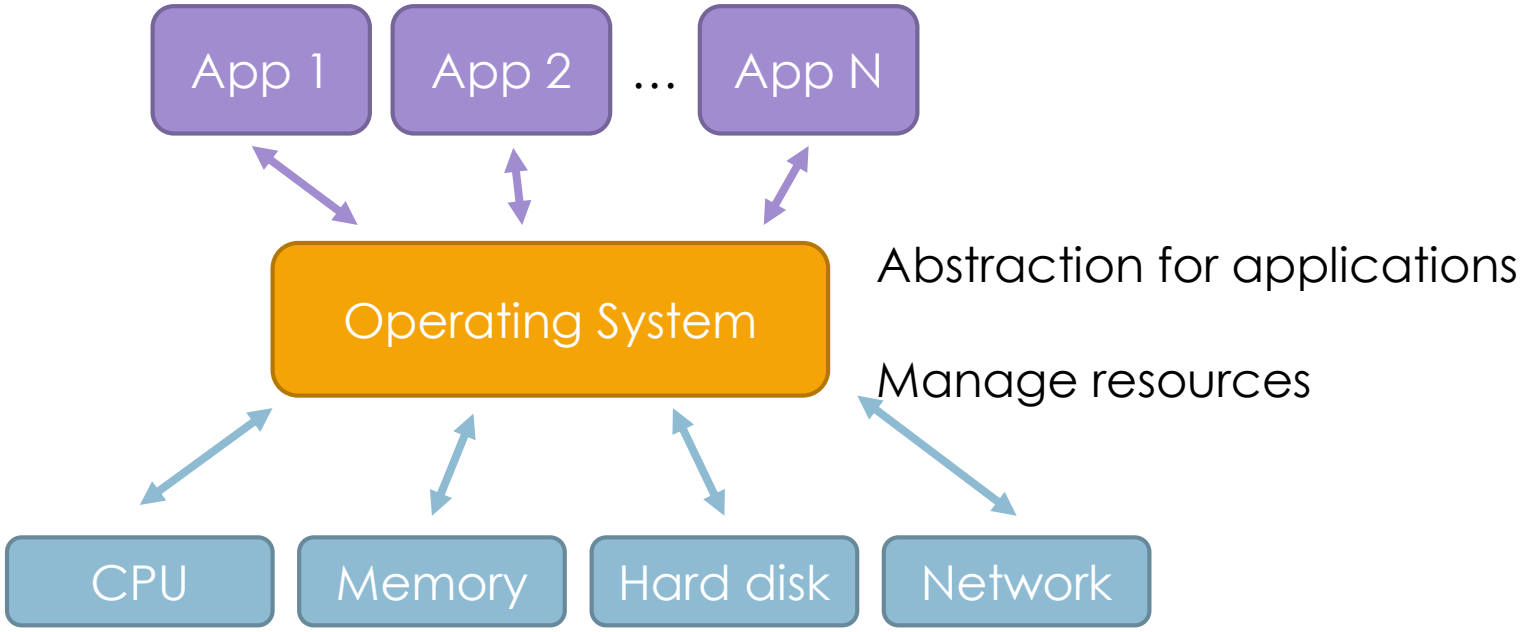
The Core of SDN

- **Distribution**
  - Global network view
- **Configuration**
  - Simplified model
- **Forwarding**
  - Generalized forwarding

Roadmap

- SDN
  - Design principles
  - SDN infrastructure

Analogy to Operating System



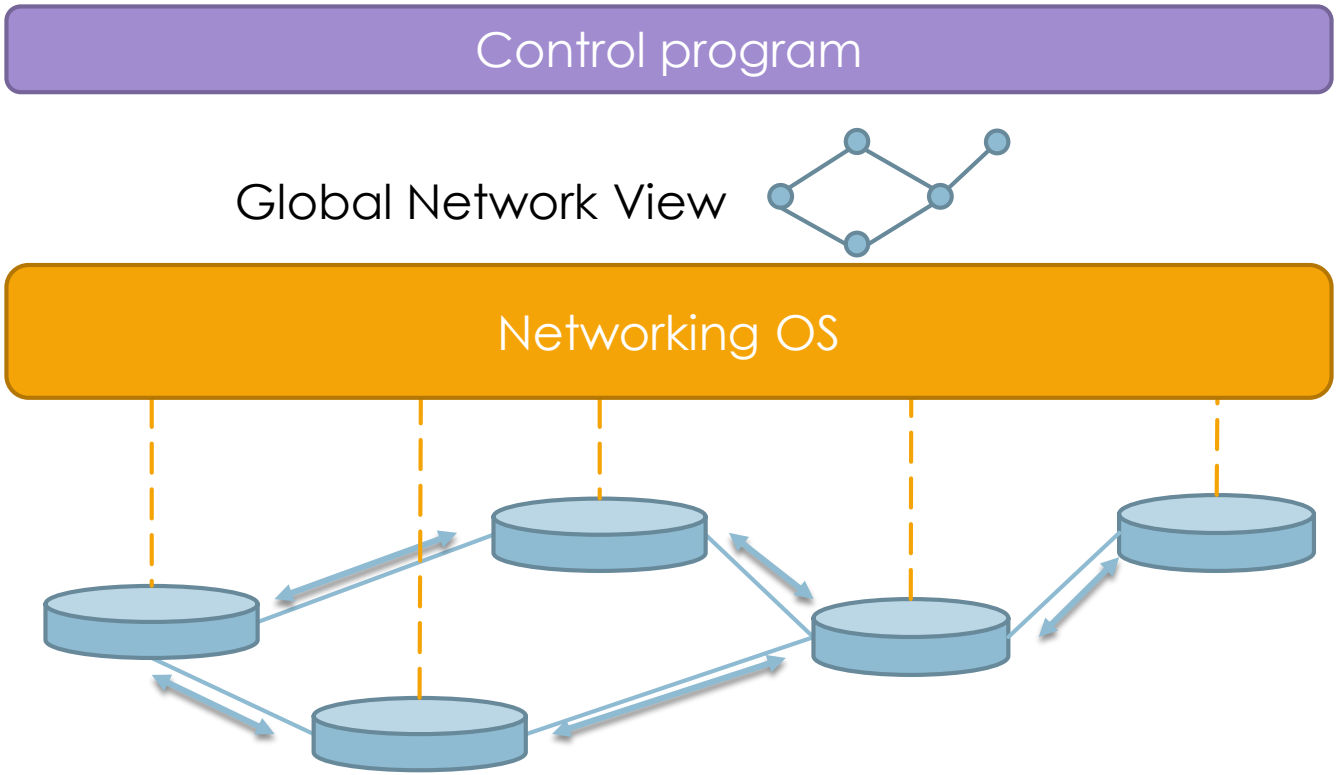
Abstraction for applications

Manage resources

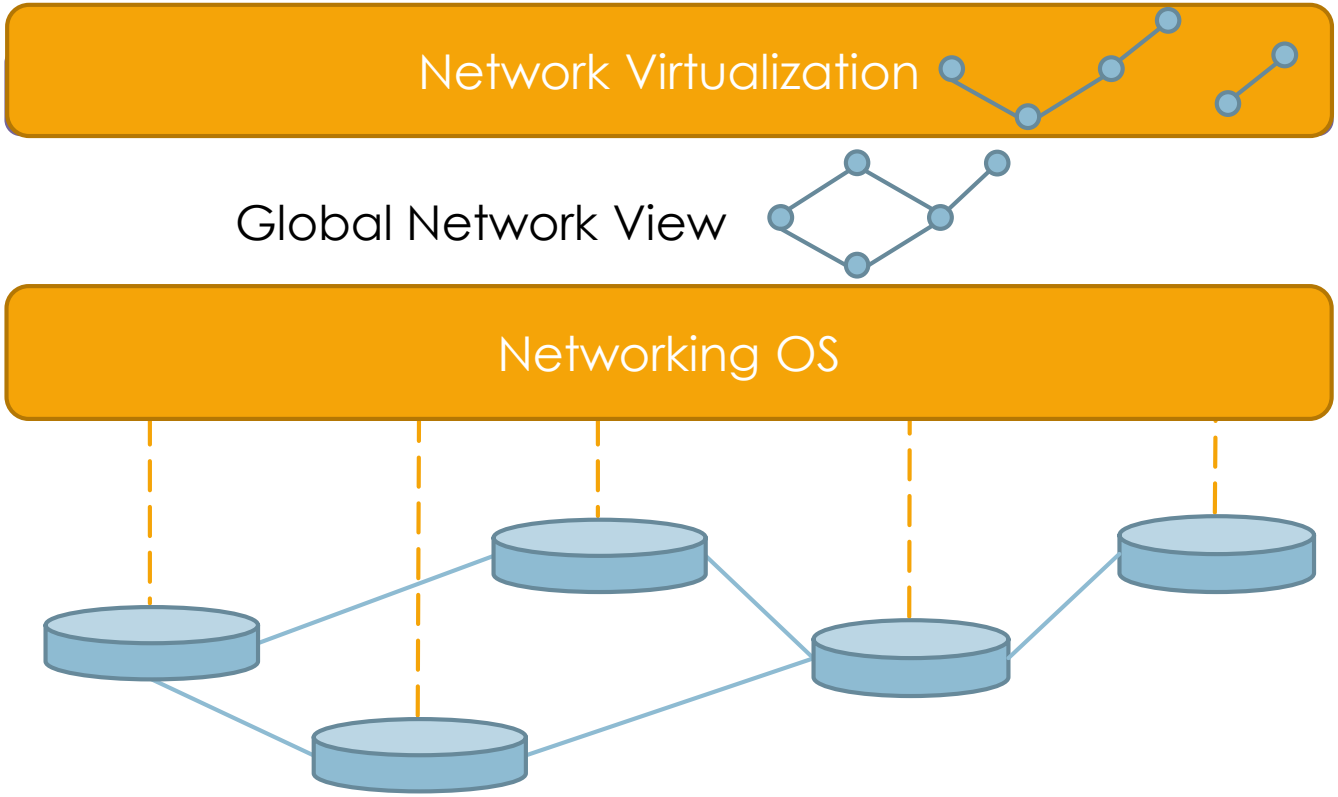
Core Challenge

- Distributed control into logically **centralized** control
  - Global network view
- Module/Layer
  - Gathers information from data plane
  - Sends commands to network devices
- Network Operating System

Distribution Layer



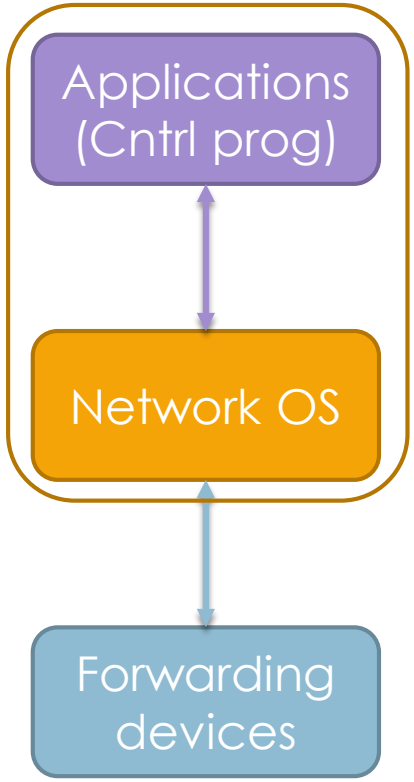
Network Virtualization



Control Plane Abstraction

- Control program
  - Specify **behavior** on simplified model
- Network virtualization
  - Global network view → **Simplified model**
  - Map the behavior back on global view
- Network OS: **logically centralized control**
  - Physical devices → **Global network view**
  - Translate to physical switches

To Be More Specific



- Network applications
  - Routing algorithm
  - Intrusion detection, etc.
- Northbound interface
- Network OS
  - Topology/inventory
  - Statistic
- Southbound interface
  - **OpenFlow**, etc.
- Forwarding devices/ data plane

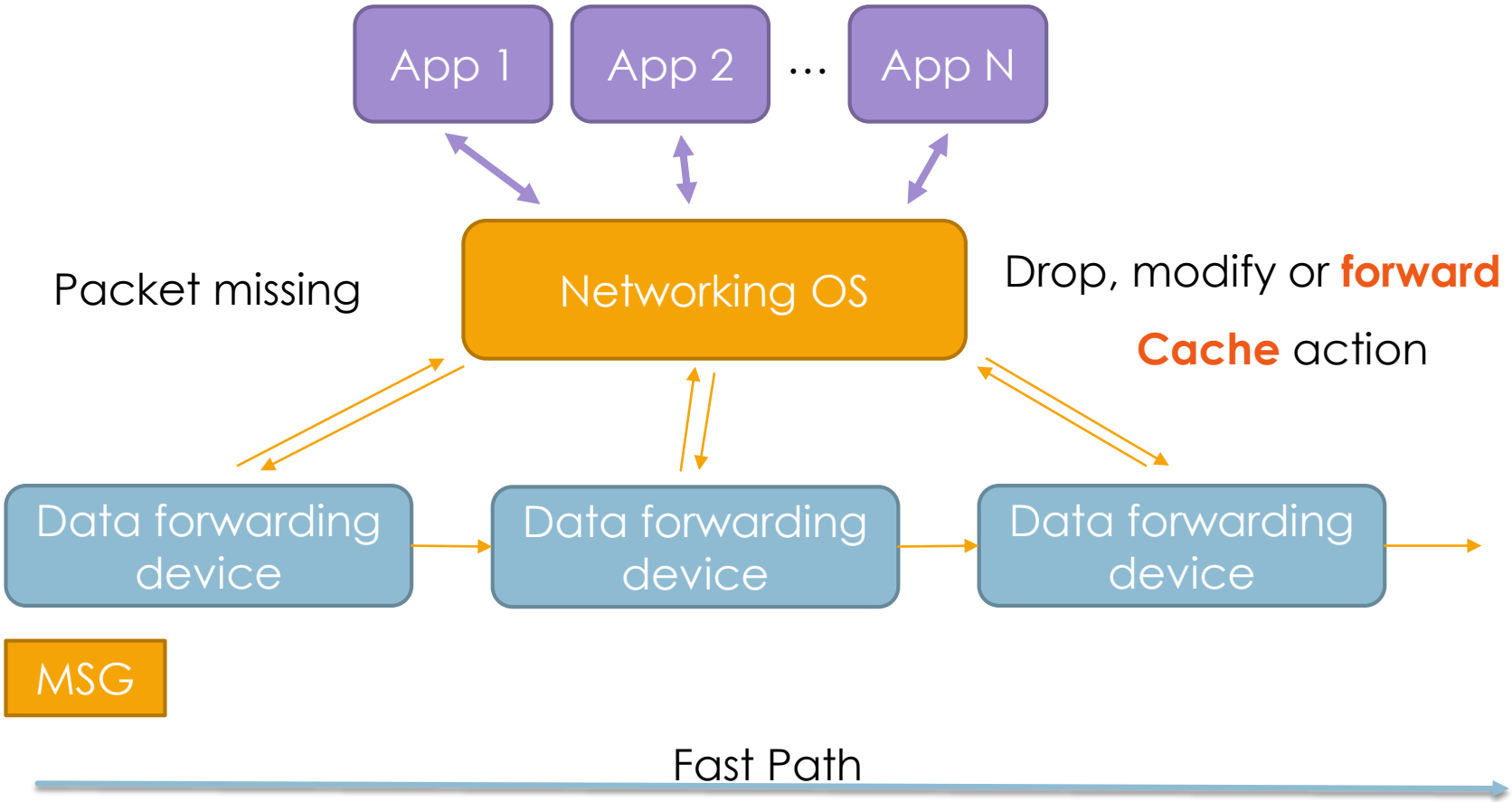
Roadmap

- SDN
  - Design principles
  - SDN infrastructure
  - OpenFlow

OpenFlow

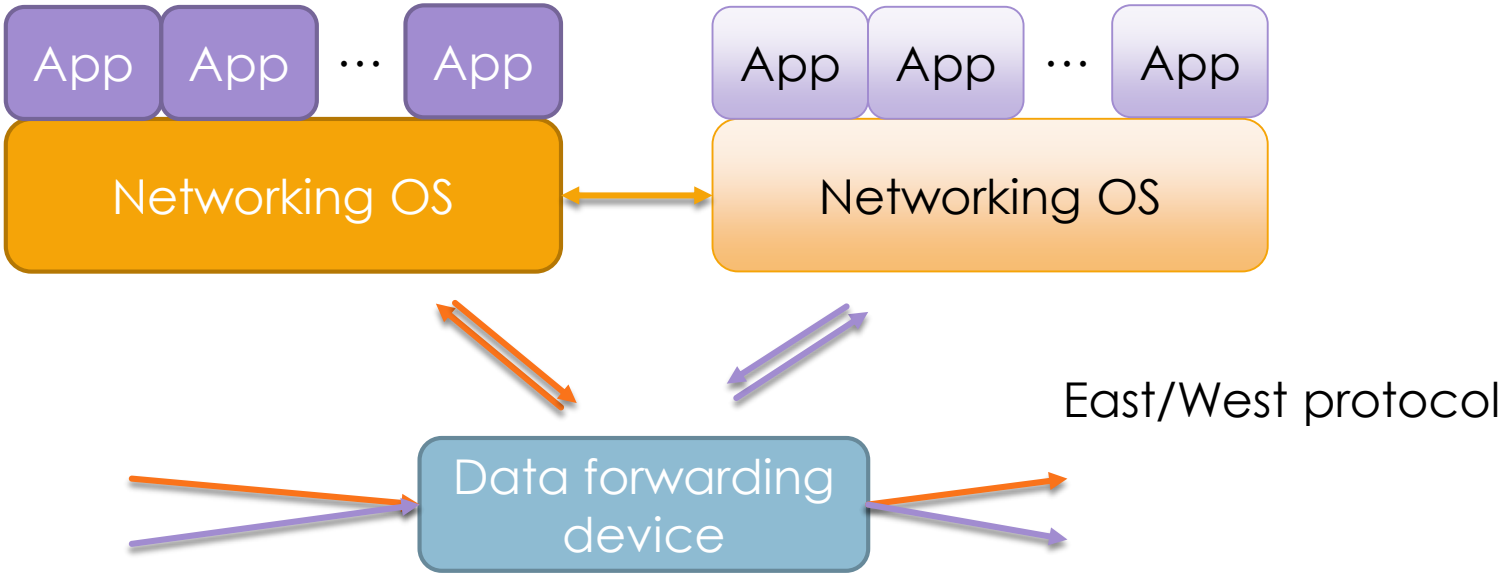
- Standard communication interface between **control** and **data** planes of SDN

Data Flow in SDN

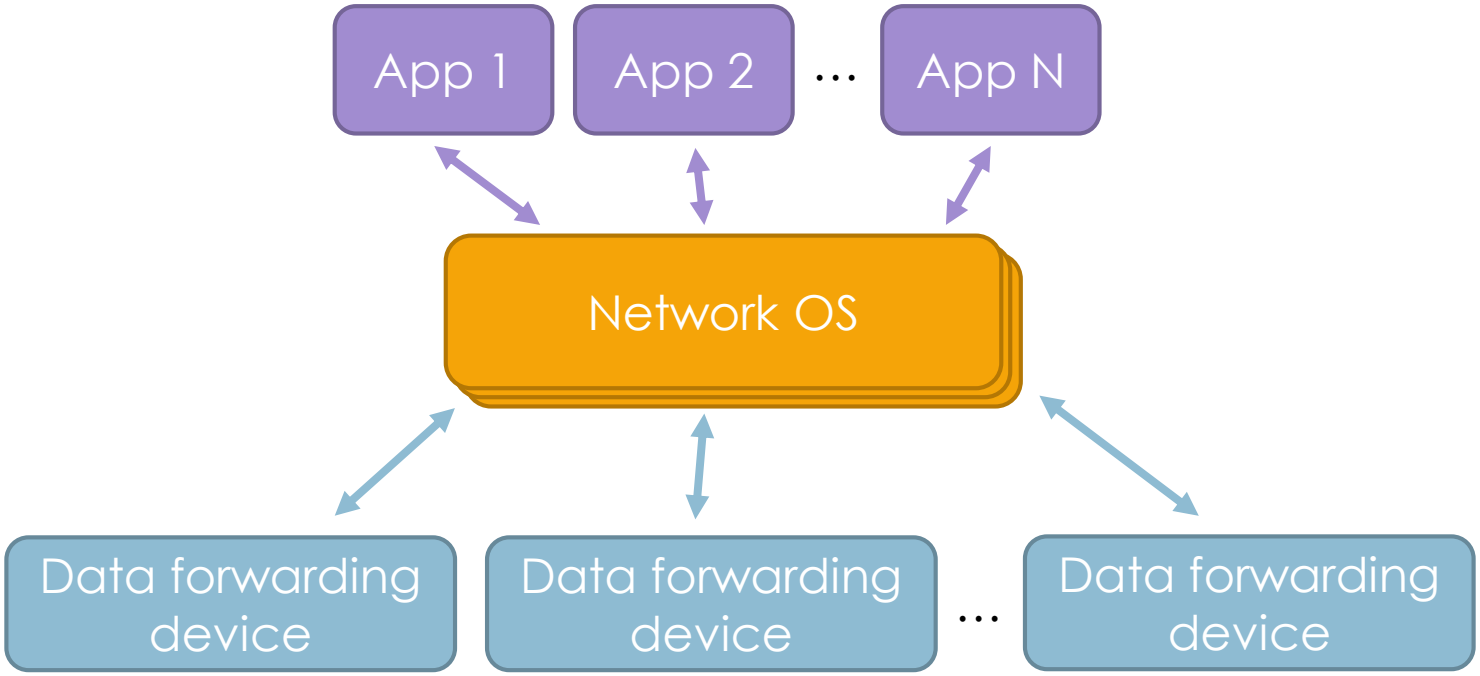


Fault-tolerance & Scalability

Logically centralized control ≠ Single physical controller



Fault-tolerance & Scalability



Roadmap

- SDN
  - Design principles
  - SDN infrastructure
  - OpenFlow
- Mininet

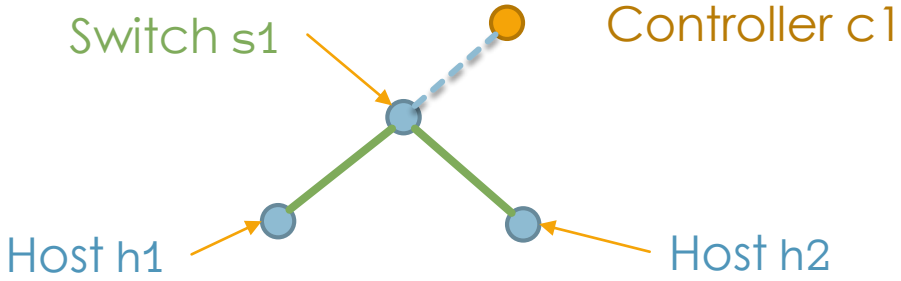
Quick Overview

- What is mininet
  - Network emulator
  - Running ***real kernel***, ***switches*** and ***application code*** on a **single machine**
- Why mininet
  - Accurate and convincing emulation
  - Building a customized network is simple and fast
  - Command-line interface (CLI) and a handy Python API

Motivation

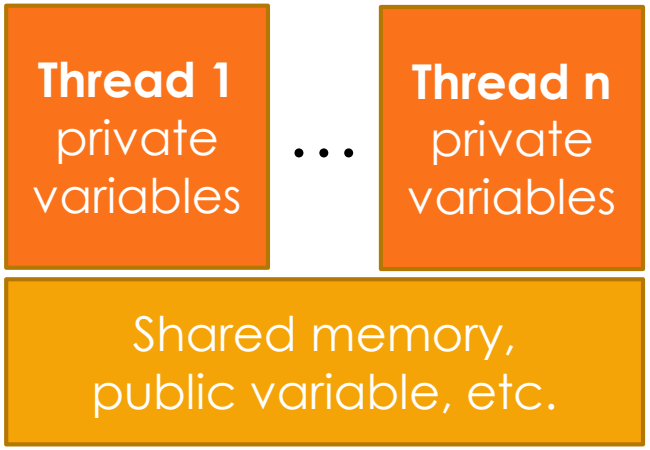
- Imagine you are deploying a cluster of tens of thousands of hosts
  - How to make sure network settings work?
- Imagine you are a poor PhD student (as I am …)
  - You want to validate your networked system in large scale setting
  - However, you only own a workstation or a laptop

Mininet Basics
build a network



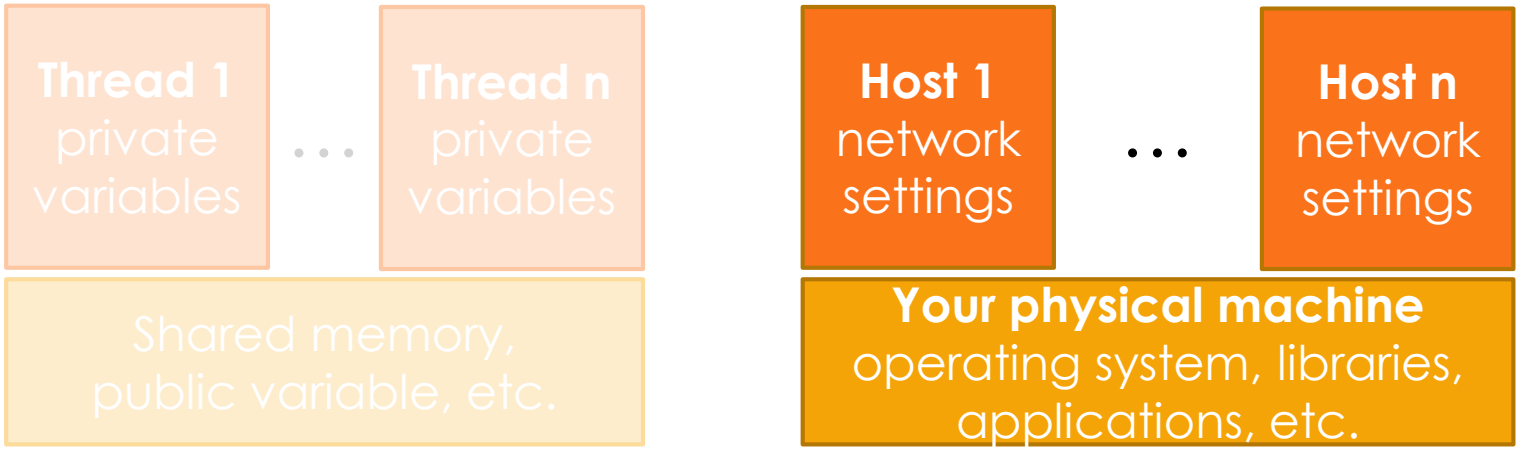
- Example: building a two-host network
  - `sudo mn`

Mininet Basics
Hosts



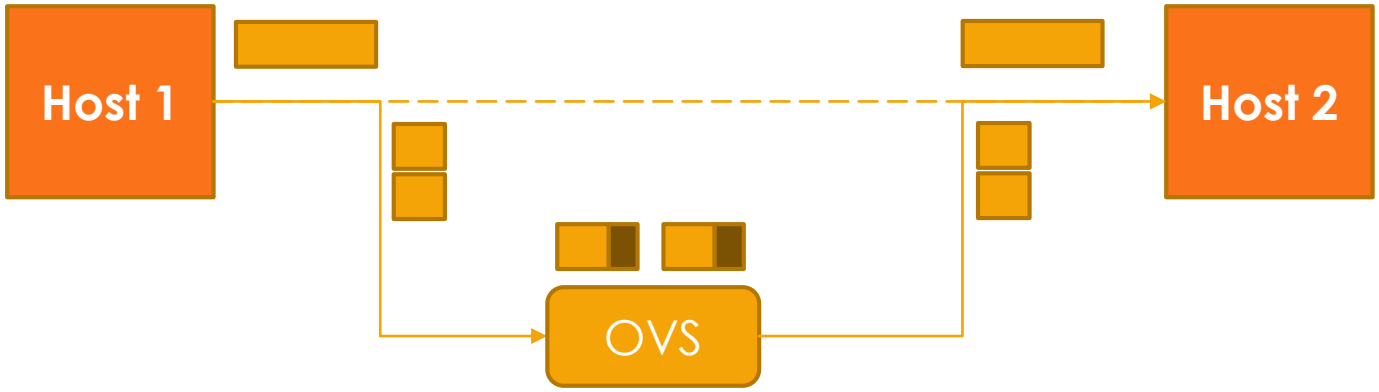
Multi-threading model

Mininet Basics
Hosts



- Light-weighted copy of your machine
  - Shares all resources of your machine
  - Except for ***network settings*** (e.g., IP, Ethernet)
  - Connected via ***open virtual switch*** (i.e., OVS)

Mininet Basics
Data Flow



- Exactly as you send through physical network
  - Execute the kernel code
  - Split the packet, encapsulate, resemble, …

Mininet Basics
Capabilities of Hosts

- Libraries, applications shared among all virtual hosts
  - ***Run your own program*** on virtual hosts!!
  - [hostname] [cmd]

Demo: nc Command

Features

- Various topologies
  - Trees, linear, etc.
- Run customized control program
- Even run a third-party controller

Python API

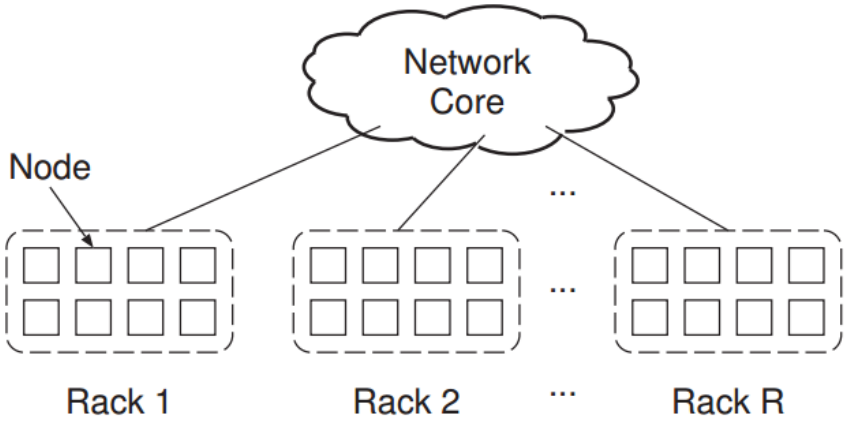
Switch
Host
Link

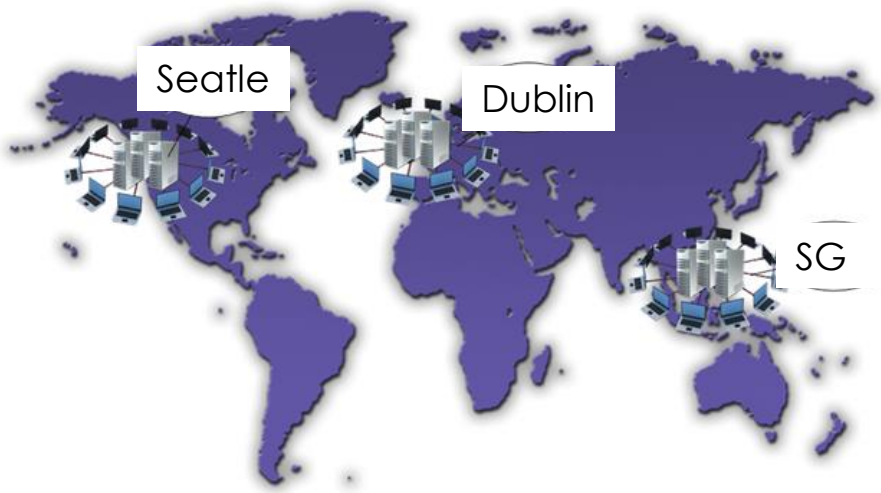
```
class SingleSwitchTopo(Topo):
  "Single switch connected to n hosts."
  def __init__(self, n=2, **opts):
    Topo.__init__(self, **opts)
    switch = self.addSwitch('s1')
    for h in range(n):
      # Each host gets 50%/n of system CPU
      host = self.addHost('h%s' % (h + 1),
                          cpu=.5 / n)
      # 10 Mbps, 5ms delay, 10% loss
      self.addLink(host, switch,
          bw=10, delay='5ms', loss=10, use_htb=True)

def perfTest():
  "Create network and run simple performance test"
  topo = SingleSwitchTopo( n=4 )
  net = Mininet( topo=topo,
      host=CPULimitedHost, link=TCLink,
      autoStaticArp=True )
  net.start()
  print "Dumping host connections"
  dumpNodeConnections(net.hosts)
  print "Testing bandwidth between h1 and h4"
  h1, h4 = net.getNodeByName('h1', 'h4')
  net.iperf( ( h1, h4 ), l4Type='UDP' )
  net.stop()

if __name__ == '__main__':
  setLogLevel('info')
  perfTest()
```

Case Study
Distributed File System Emulator





Network is the major bottleneck!

Conclusion

- Control problem in traditional network
  - Lack of abstraction
- **Distribution**, **configuration** and **forwarding**
- **NOS**, **network virtualization** and **general forwarding**
- SDN infrastructure
- Mininet: handy but powerful network emulator

Roadmap

- SDN
  - Design principles
  - SDN infrastructure
  - OpenFlow
- Mininet
- Lessons learned

"The Power of Abstration"

"Modularity based on abstraction is the way things get done"

-Barbara Liskov

Abstraction → Interfaces → Modularity

Extracting Simplicity

- The ability to **master complexity** is not the same as the ability to **extract simplicity**
- When first getting systems to work
  - Mastering complexity
- When making systems easy to use and understand
  - Extracting simplicity
- **You will never succeed in extracting simplicity**
  - Unless realize it is different from mastering complexity

References

- Kreutz, et al. **"Software-Defined Networking: A Comprehensive Survey"**, IEEE Trans. on Computers
- Scott Shenker, **"The Future of Networking and the Past of Protocols"**
- Mininet [http://mininet.org/] tutorials:
  - https://www.youtube.com/channel/UCEoaojfEY_6L5TWWjIn9t9Q